МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет економіки, підприємництва та інформаційних технологій

Кафедра інформаційних технологій

Допущено до захисту

"_____" _____ 2026 р.

Зав. кафедри, к.т.н., доцент

_____ **Ліщук Р.І.**

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ УПРАВЛІННЯ ПРОЄКТАМИ ТА ПРОЦЕСАМИ З РОЗШИРЕНИМ ФУНКЦІОНАЛОМ

Виконав – студент 4 курсу, групи 41-кн

освітньої програми "Комп’ютерні науки"

спеціальності 122 Комп’ютерні науки

Горбенко Михайло Васильович

Науковий керівник – д.т.н., професор

Кучерук Володимир Юрійович

Умань – 2026

ЗМІСТ

Перелік умовних позначень, скорочень і термінів**Помилка! Закладку не визначено.**

ВСТУП 3

РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ПРОГРАМНИХ ЗАСОБІВ УПРАВЛІННЯ ПРОЕКТАМИ 5

1.1 Огляд існуючих систем управління проектами 5

1.2 Огляд існуючих концепцій 8

1.3 Концепція MEAN 11

РОЗДІЛ 2. ПРОЕКТУВАННЯ СТРУКТУРИ ТА КОМПОНЕНТІВ СИСТЕМИ 15

2.1 Планування реалізації операцій CRUD 15

2.2 Проектування та методи розробки компонентів системи 17

2.3 Проектування інтерфейсу додатку 21

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ПРОЕКТАМИ 25

3.1 Підготовка середовища для розробки 25

3.2 Розробка компонентів для обробки запитів та інтерфейсу 30

3.3 Тестування програмного забезпечення 41

ВИСНОВКИ 50

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 52

ДОДАТКИ ..... **Помилка! Закладку не визначено.**

Додаток А. Серверна частина (Backend: Node.js + Express + MongoDB)

.....**Помилка! Закладку не визначено.**

ВСТУП

Нині створюється дедалі більше різноманітних проектів, які потребують чіткого та впорядкованого керівництва, тому спеціалізований додаток стає незамінним інструментом. Такі системи дозволяють менеджерам або керівникам легко відстежувати статус конкретних завдань чи всієї роботи загалом, оперативно змінювати етапи процесів і призначати виконавців. Грамотне управління є критично важливим для досягнення очікуваного результату, оскільки воно забезпечує рівномірний розподіл навантаження між працівниками, а також допомагає точно розрахувати час і кошти, необхідні для реалізації цілей. Сучасні технології у відкритому доступі дають можливість кожному створити подібну систему. Зараз особливо актуальним є створення вебдодатків за допомогою мови програмування TypeScript, створеної на базі JavaScript. Зокрема, використання фреймворку Angular дозволяє будувати структуровані й типізовані рішення з окремих невеликих компонентів, що значно оптимізує та прискорює весь процес розробки.

Метою дослідження є розширення функціоналу програмного забезпечення для управління проектами та процесами, а також збільшення можливостей для подальшої модифікації та масштабування продукту. Для реалізації цієї мети необхідно послідовно виконати кілька ключових завдань, які включають аналіз сучасних програмних засобів та огляд існуючих аналогів у сфері менеджменту. Також потрібно спроектувати надійну структуру та компоненти системи, розробити безпосередньо сам додаток для одночасної роботи кількох користувачів, провести детальне тестування створеного забезпечення та підготувати інструкцію для користувача. Об'єктом дослідження виступає процес управління проектами, тоді як предметом є саме програмне забезпечення для цього управління. У ході роботи застосовуються методи аналізу даних, розробки автоматизованих тестів, теорії

алгоритмів та оптимізації. Технологічна основа розробки базується на поняттях клієнт-серверної архітектури, використанні формату обміну даними JSON та обов'язковому шифруванні даних користувачів для гарантування належного рівня безпеки. Практична цінність отриманих результатів полягає у створенні готового програмного засобу із сучасною архітектурою, який оптимізує взаємодію всередині команди, при цьому закладені в систему алгоритми мають потенціал для подальшого вдосконалення відповідно до появи нових технологічних рішень.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ПРОГРАМНИХ ЗАСОБІВ УПРАВЛІННЯ ПРОЕКТАМИ

1.1 Огляд існуючих систем управління проектами

Програмне забезпечення Jira, створене австралійською компанією Atlassian у 2002 році як інструмент відстеження проблем для розробників, згодом перетворилося на популярну систему управління проектами завдяки поширенню методології Agile. Зі зростанням популярності платформа розширилася і наразі пропонує кілька спеціалізованих внутрішніх продуктів для різних команд. Версія Jira Core функціонує як загальна система управління проектами для різних сфер діяльності. Варіант Jira Software адаптований спеціально для розробників і містить робочі процеси для планування та випуску програмного продукту на основі методологій Scrum і Kanban, де основними типами проблем є Bug для виправлення помилок, Story для впровадження нових можливостей та Task для інших завдань. Для IT-команд, служб підтримки та кол-центрів створена надбудова Jira Service Desk, яка допомагає керувати квитками, інцидентами та змінами.

З технічного боку Jira написана на мові програмування Java і розгортається як стандартний файл Java WAR у контейнері Servlet Java, наприклад Tomcat. Для виклику віддалених процедур у системі використовується REST, тоді як від підтримки SOAP та XML-RPC відмовилися починаючи з версії 7.0. Оскільки Jira є вебдодатком, користувачі взаємодіють із нею через браузер. Для обробки вебзапитів застосовується фреймворк MVC WebWork 1, схожий на Struts, де кожен запит обробляється певною дією з використанням класів утиліт та менеджерів. За генерацію більшої частини HTML, який користувач бачить у відповідь на свій запит, відповідає JSP, що використовується для шару View.

Іншим популярним інструментом є Trello, призначений для спільного керування роботою, відстеження проектів, розподілу завдань та візуалізації прогресу. Ця програма спирається на принципи проектних дощок Kanban, що

забезпечує легкий і зрозумілий огляд усього процесу. Головна перевага Trello полягає в його простоті та універсальності, завдяки чому додаток часто використовують навіть для особистих цілей, наприклад для планування свят чи подорожей. Це відрізняє Trello від складного повнофункціонального програмного забезпечення, оскільки сервіс надає перевагу доступності та легкій функціональності перед занадто широким набором функцій.

Програма Trello, яку у 2017 році придбав постачальник програмного забезпечення Atlassian, відома своєю здатністю успішно інтегруватися з іншими додатками за допомогою спеціальних посилань. Такі інтеграції API дозволяють користувачам взаємодіяти з функціями сервісів Slack, Salesforce, Gmail та інших платформ, включаючи власні продукти Atlassian, безпосередньо з інтерфейсу Trello. На початку свого створення проект розроблявся як чистий JavaScript на клієнтській та серверній частинах, проте згодом розробники перевели весь код на мову CoffeeScript. Зараз клієнтська частина платформи використовує такі технології, як Backbone.js, HTML5 та Mustache. У свою чергу серверна частина написана на Node.js із застосуванням інструментів HAProxy та Redis, а для надійного збереження всієї інформації використовується база даних MongoDB.

Програмне забезпечення Microsoft Project призначене для ефективного управління проектами, створення детальних розкладів, планів, обліку часу та керування ресурсами. Для професійної роботи в цій системі використовуються такі інструменти, як діаграми Ганта, дошки Kanban та спеціальні календарі. Цей продукт також відомий під назвами MS Project або Project Professional, що є його поточною офіційною назвою. Окрім цього, існують інші споріднені рішення, серед яких Project Online, Project Server та Project для Інтернету, доступ до яких можна отримати шляхом придбання підписки Microsoft Project Plan з одним із трьох рівнів цін на вибір. Історія цього продукту розпочалася з додатку для MS-DOS, який спочатку написали на мові програмування C та мові асемблера для комп'ютерів IBM PC. Сама ідея виникла на початку вісімдесятих років минулого століття у Рона

Бредехофта, який вирішив продемонструвати принципи менеджменту на прикладі рецепту та підготовки сніданку з яєць Бенедикт. Під час створення програми він сформував компанію Microsoft Application Services, яка згодом уклала OEM угоду з корпорацією Microsoft. Менеджер з розробки продуктів Microsoft Алан Бойд представив цю програму як внутрішній інструмент для керування великою кількістю поточних програмних проектів усередині корпорації, написав технічну специфікацію та залучив сторонню фірму із Сіетла для створення першого прототипу.

Програма Microsoft Project створює бюджети на основі виконання запланованих завдань та встановлених ставок ресурсів. Під час розподілу людей, обладнання чи матеріалів за конкретними завданнями та оцінювання обсягу роботи система автоматично обчислює вартість, яка дорівнює помноженій на роботу ставці, після чого ці показники послідовно підсумовуються на рівні завдань, підсумкових етапів і всього проекту загалом. Усі визначені ресурси можна спільно використовувати між різними планами за допомогою єдиного пулу ресурсів. При цьому кожен ресурс може мати індивідуальний календар із зазначенням доступних днів та змін, а його фінансові ставки слугують для розрахунку витрат на призначення, які згодом згортаються на рівні ресурсів. Програма дозволяє призначати один ресурс на кілька завдань у різних планах, як і одному завданню надавати кілька виконавців, формуючи загальний розклад на основі доступності, зафіксованої в календарях. Проте всі матеріальні ресурси визначаються в етикетці без обмежень, через що система не здатна вирахувати точну кількість готової продукції із заданого обсягу сировини. Така особливість робить дане програмне забезпечення непридатним для вирішення завдань виробництва, обмеженого наявними матеріалами, тому для управління складними підприємствами з випуску фізичних товарів необхідне додаткове спеціалізоване забезпечення.

У сфері планування додаток створює розклади критичних шляхів, а за допомогою сторонніх розширень підтримує методології критичного ланцюга та

ланцюга подій. Усі графіки можна вирівнювати за ступенем завантаженості виконавців, а самі ланцюжки завдань наочно візуалізуються на діаграмі Ганта. Крім того, Microsoft Project розпізнає різні класи користувачів, надаючи їм диференційовані рівні доступу до проектів, представлень та іншої конфіденційної інформації. Усі налаштовані користувачем об'єкти, включаючи календарі, таблиці, фільтри, поля та представлення даних, надійно зберігаються в корпоративній глобальній мережі, яка є спільною та доступною для всіх учасників робочого процесу.

1.2 Огляд існуючих концепцій

Система управління проектами має відповідати низці технічних та функціональних вимог. При першому ознайомленні з платформою має підтримуватись реєстрація нового користувача зі збереженням його даних та їх подальшим шифруванням у вигляді токена. Для вже зареєстрованих користувачів передбачена можливість авторизації через запит до сервера, після успішної автентифікації на клієнтській частині додатка користувач отримує свої дані та зберігає їх у зашифрованому вигляді в local storage, яке є вбудованим сховищем у браузері. Реалізація форми реєстрації відбуватиметься за допомогою formGroup на основі технології reactive forms, що забезпечує зручне керування та маніпулювання полями введення.

Після успішного входу клієнтська частина додатка автоматично відправляє запит на сервер для отримання списку проектів, де користувач є учасником або менеджером. Отримані проекти зберігатимуться у вигляді масиву безпосередньо в компоненті, який відповідає за цей розділ. На інтерфейсі цей список відображатиметься у вигляді карток з базовою інформацією про кожен проект. Для переходу до управління конкретним проектом користувачеві достатньо натиснути на його картку, після чого спрацює функція click, яка перенаправить його на сторінку з детальною інформацією, поточними завданнями та доступними діями

всередині проекту.

Загальна концепція сучасного фреймворку являє собою готову інфраструктуру певних програмних рішень, за допомогою яких створення складних систем суттєво полегшується. Такі інструменти містять набір базових рішень, бібліотек та архітектурних налаштувань, які оптимізують розробку як малих сервісів, так і великих комплексних платформ.

Платформа розробки та дизайну додатків Angular призначена для створення ефективних і складних односторінкових програм. Цей фреймворк використовує як основну мову програмування TypeScript і суворо притримується концепції об’єктно-орієнтованого програмування, оскільки кожен елемент його інфраструктури є інкапсульованим об’єктом, який можна повторно застосовувати в будь-якому місці програми. Натомість React.js є бібліотекою JavaScript з відкритим кодом, яку розробив інженер компанії Facebook Джордан Волке для побудови інтерфейсів користувача в односторінкових застосунках. Цей інструмент відповідає за обробку шаблонів у веб та мобільних додатках, а також дозволяє створювати багаторазові компоненти інтерфейсу. Історія практичного розгортання React розпочалася зі стрічки новин Facebook у 2011 році та платформи Instagram у 2012 році. Своєю чергою фреймворк AngularJS, що базується на JavaScript, надає розробникам набір декларативних шаблонів, механізми впровадження залежностей та наскрізні інструменти, які полегшують проектування інтернет-ресурсів та вирішують типові проблеми веб-розробки. Ця популярна технологія активно використовується в багатьох загальнодоступних сервісах, включаючи Google Cloud Platform і AdWords, а також у внутрішніх інструментах самої компанії Google.

Фреймворк JavaScript з відкритим вихідним кодом Vue.js, створений Еваном Ю та підтримуваний активною спільнотою, призначений для проектування інтерфейсів користувача та односторінкових застосунків. Його особливістю є поступово адаптивна архітектура, де основна бібліотека фокусується виключно на шарі перегляду, а складні функції на кшталт маршрутизації, керування станом та

інструментів збирання реалізуються через офіційні допоміжні пакети. Vue.js також дозволяє розширювати HTML за допомогою вбудованих або користувацьких директив, які додають нові функціональні можливості. У сфері серверної розробки виділяється Laravel, який є надійним PHP-фреймворком з виразним синтаксисом. Він базується на популярному шаблоні проектування модель-представлення-контролер та, за даними Google Trends, утримує позицію найпотужнішого інструменту у своєму сегменті, пропонуючи стандартизовану й багатофункціональну платформу для високопродуктивних вебдодатків.

Для розробки серверної частини також активно застосовується Node.js, що є міжплатформним середовищем виконання JavaScript з відкритим кодом. Завдяки запуску движка V8 поза браузером цей інструмент демонструє надзвичайно високу продуктивність. Програми на базі Node.js функціонують в одному процесі без створення окремого потоку під кожен новий запит. Завдяки набору асинхронних примітивів введення-виведення у стандартній бібліотеці та неблокуючим парадигмам, блокування коду тут є рідкісним винятком. Під час роботи з мережею, базами даних або файловою системою Node.js не витрачає цикли процесора на очікування, а просто відновлює операцію після отримання відповіді. Такий підхід дозволяє успішно обробляти тисячі одночасних підключень на одному сервері й уникати складних помилок, пов'язаних із паралельністю потоків.

Фреймворк Express.js є базовою структурою вебдодатків для Node.js, яка розповсюджується як безкоштовне програмне забезпечення з відкритим вихідним кодом під ліцензією MIT. Цей інструмент розроблений для побудови вебдодатків та API, і його часто називають де-факто стандартною серверною платформою у своїй екосистемі. Оригінальний автор проекту TJ Holowaychuk описував своє творіння як сервер, натхненний Sinatra, що вказує на його відносну мінімалістичність, де більшість додаткових функцій реалізуються за допомогою плагінів. Express.js виступає обов'язковим внутрішнім компонентом таких популярних стеків розробки, як MEAN, MERN або MEVN, де він працює разом із

базою даних MongoDB та відповідною інтерфейсною структурою чи бібліотекою JavaScript. Нижче наведено детальний аналіз та зіставлення характеристик трьох найпопулярніших інструментів, що використовуються для проектування сучасних користувацьких інтерфейсів.

Таблиця 1.1 – Порівняльна характеристика фреймворків та бібліотек для розробки користувацького інтерфейсу

Характеристики	Angular2+	React.js	Vue.js
Тип	Фреймворк	Бібліотека	Фреймворк
Особливості	Фреймворк містить в собі всі інструменти, необхідні для розробки користувацьких інтерфейсів (роутинг, модулі, інтернаціолізацію, тощо). Для стилів, розмітки та функціоналу різні файли, що полегшує орієнтацію в компонентах	Бібліотека, що містить основні модулі, але з ними не можна побудувати складний інтерфейс, також поєднує мову розмітки та функціональний код, що може бути важким для сприйняття	Фреймворк, що містить в собі основні модулі для розробки інтерфейсу, об'єднує стилі, розмітку та функціональний код в одному файлі, що не завжди дуже зручно
Мова	TypeScript	TSX або JSX	JavaScript
Швидкість освоєння	Доволі складний фреймворк, який потребує часу на його вивчення, але в підсумку є дуже потужним інструментом	Середня по складності бібліотека, якщо не брати до уваги допоміжні модулі, які доволі важко вивчаються	Найлегший для освоєння фреймворк
Сфера застосування	користовується для розробки інтерфейсу середніх та великих проектів	користовується для розробки інтерфейсу проектів невеликих та середніх масштабів	користовується для розробки інтерфейсу середніх та великих проектів
Можливості	Можна розробляти веб-інтерфейси для будь якої платформи, та з будь-яким функціоналом, адже це інструмент надзвичайно потужний з коробки	Можна розробляти інтерфейси для будь-яких платформ, але функціонал, що йде з коробки, не дозволяє розробити багатофункціональні інтерфейси	Можна розробляти інтерфейси для будь-якої платформ, функціонал, що йде з коробки дозволяє робити функціональні та складні інтерфейси

1.3 Концепція MEAN

Концепція MEAN представляє собою сучасний стек розробки вебдодатків, який базується на чотирьох ключових технологіях, де MongoDB виступає базою даних для збереження інформації, Express.js слугує фреймворком для полегшеної

побудови функціоналу та інтерфейсу серверної частини, Angular відповідає за створення клієнтської частини разом із користувацьким інтерфейсом, а Node.js є рушієм для роботи сервера. Об'єднання цих інструментів є надзвичайно вдалим, оскільки практично кожен із них використовує мову програмування JavaScript. Винятком є фреймворк Angular, який працює на TypeScript, проте різниця між ними мінімальна, адже TypeScript є прямим продовженням та вдосконаленням JavaScript, збагаченим суворою типізацією, інтерфейсами для об'єктів та іншими корисними функціями.

Сама концепція Angular є простою для розуміння та вивчення, оскільки кожен елемент на сторінці може виступати окремим незалежним компонентом зі своїми власними стилями, методами та каркасом. Розробник має можливість вільно використовувати такі компоненти в будь-якому місці системи, а їх видалення чи переміщення в іншу частину додатка вимагає мінімальних змін у коді й не порушує загальну працездатність платформи. Завдяки інкапсульованим стилям зміна зовнішнього вигляду одного компонента ніяк не впливає на інші елементи, а для корекції загального дизайну використовуються окремі файли з глобальними стилями. Теоретично Angular-програми можна створювати, прописуючи весь код на одній сторінці всередині однієї величезної функції, проте такий підхід абсолютно неефективний з точки зору структурування коду і повністю позбавляє сенсу саме існування архітектури Angular.

В архітектурі Angular широко використовується концепція модулів, де кожен модуль є фрагментом коду, створеним для однієї конкретної мети, а сам додаток повністю збирається з таких елементів, частина з яких може застосовуватися лише в одному місці, а інша в різних частинах програми. Шаблони поєднують стандартний HTML з фірмовою розміткою фреймворку, яка здатна трансформувати елементи перед їх виведенням на екран. При цьому директиви шаблонів забезпечують логіку програми, а розмітка зв’язування з’єднує дані програми та DOM за допомогою двох існуючих типів прив’язки даних.

Для реалізації логіки або збереження даних, які безпосередньо не пов’язані з конкретним інтерфейсом і мають використовуватися різними компонентами спільно, розробники створюють класи служб. Визначення такого класу обов'язково супроводжується декоратором Injectable, який надає метадані для механізму ін’єкції залежностей. Цей механізм DI дозволяє зберігати класи компонентів компактними та ефективними, оскільки вони не займаються прямим отриманням даних із сервера, валідацією введення або логуванням в консоль, а делегують подібні завдання відповідним службам.

У свою чергу модуль Angular Router NgModule надає сервіс для визначення навігаційних шляхів між різними станами програми та ієрархіями переглядів. Його робота повністю змодельована на основі звичних умов браузерної навігації, як-от введення URL в адресний рядок, перехід за посиланням або використання кнопок назад і вперед для переміщення по історії відвіданих сторінок. Проте маршрутизатор відображає подібні до URL шляхи не до нових сторінок, а до конкретних переглядів. Коли користувач робить дію, яка зазвичай завантажує нову сторінку, маршрутизатор перехоплює стандартну поведінку браузера та просто динамічно показує чи приховує необхідні ієрархії перегляду.

Якщо маршрутизатор визначає, що поточний стан програми вимагає певної функціональності, яка міститься в ще не завантаженому модулі, він може виконати ліниве завантаження цього модуля безпосередньо на вимогу. Для налаштування правил навігації розробник пов’язує відповідні шляхи зі своїми компонентами. Кожен шлях використовує синтаксис, подібний до URL-адреси, який успішно інтегрує дані програми подібно до того, як синтаксис шаблону об'єднує інтерфейс із бізнес-логікою. Це дозволяє застосовувати логіку додатка для динамічного відображення або приховування конкретних переглядів залежно від дій користувача та встановлених правил доступу.

Фреймворк Express.js є невеликою структурою, яка розширює базові можливості вебсервера Node.js для спрощення створення API та додавання нових

корисних інструментів. Він суттєво полегшує організацію внутрішньої логіки програми за допомогою маршрутизації та проміжного програмного забезпечення, збагачує стандартні HTTP об'єкти Node.js зручними утилітами, а також спрощує візуалізацію динамічного вмісту. Простими словами, цей інструмент використовують для структурування коду, виконання простої маршрутизації, підтримки технологій Connect та підключення численних сторонніх розширень.

Після виконання стандартної команди встановлення через менеджер пакетів у проекті з'являється директорія node_modules, яка містить усі залежності, необхідні для повноцінного запуску Express.js. При цьому фреймворк є ліберальним за своєю природою і не нав'язує розробнику фіксовану структуру папок, що має свої переваги та недоліки. З одного боку, це є великим плюсом для масштабних додатків, оскільки архітектори не обмежені жорсткими рамками під час проектування системи, проте з іншого боку, така свобода стає мінусом для новачків, які тільки знайомляться з Express.js і не знають, як правильно організувати файли своєї програми.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СТРУКТУРИ ТА КОМПОНЕНТІВ СИСТЕМИ

2.1 Планування реалізації операцій CRUD

Коли ми будуємо API, ми матимемо моделі, які забезпечують чотири основні типи функціональності. Модель повинна бути здатною до створення, читання, оновлення та видалення даних, які розробники часто називають загальновідомим акронімом CRUD. Модель повинна мати здатність виконувати щонайбільше ці чотири функції, щоб вважатися повною. Якщо певну дію неможливо чітко описати однією з цих чотирьох базових операцій, то вона потенційно має бути виділена у власну окрему модель.

Парадигма CRUD є надзвичайно поширеною при створенні різноманітних веб-додатків, оскільки вона забезпечує зрозумілу та чітку структуру для нагадування розробникам про те, як створювати повноцінні та зручні моделі даних. Як наочний приклад можна розглянути стандартну модель користувача. У будь-якому веб-додатку обов'язково буде присутня база даних із користувачами та працівниками, в якій надійно зберігатимуться відповідні об’єкти користувачів. Припустимо, що типовий об’єкт такого користувача має наступний вигляд.

“користувач”: {"id": <Integer>, “ім’я”: <String>, “прізвище”: <String>, “пошта”: <String>}

Для реалізації операції створення нових ресурсів у архітектурному стилі REST найчастіше застосовується метод HTTP POST. Цей метод відповідає за генерацію та додавання нового об'єкта конкретного типу до вже існуючої системи. Як приклад можна уявити ситуацію, коли розробник створює абсолютно нового користувача та вносить його до загального збереженого списку в додатку, де всі подібні профілі працівників акумулюються у відповідній базі даних. Якщо виникає потреба зареєструвати та додати такий новий елемент, клієнтська частина платформи відправляє на сервер саме запит POST.

Запит:
POST http://localhost:3399/users/
Тіло запиту -

```
{
"user": {
"first_name": “Jonh”,
"second_name": “Levko”,
"email": “Levko@gmail.com”,
"password": “pass111”,
}
}
```

Ця дія створює в базі даних новий елемент, який містить ім'я John, прізвище Levko, електронну пошту Levko@gmail.com та пароль pass111. У разі успішного завершення операції сервер має надіслати у відповідь спеціальний заголовок із посиланням на цей щойно створений ресурс, а також повернути стандартний код стану HTTP 201 із повідомленням про успішне створення. Для операції читання та отримання даних у архітектурі REST застосовується метод GET. Важливою умовою є те, що читання ресурсу за жодних обставин не повинно змінювати чи модифікувати інформацію на сервері, воно призначене виключно для її вилучення. Завдяки цьому принципу ідемпотентності, якщо надіслати запит GET до одного й того самого ресурсу десять разів поспіль, відповідь під час першого та останнього викликів буде абсолютно однаковою. Цей метод можна використовувати як для отримання конкретного об'єкта, так і для перегляду всього переліку елементів, наприклад, через адресу http://localhost:3399/users/ користувач зможе отримати повний список усіх зареєстрованих у системі людей. У свою чергу для оновлення та редагування наявної інформації в рамках парадигми CRUD застосовується метод HTTP PUT. Як приклад, якщо один із користувачів вирішить змінити свій старий пароль на новий, розробнику необхідно внести корективи до бази даних та оновити відповідне поле, що й реалізується за допомогою надсилання запиту PUT.

```
Запит: PUT http://localhost:3399/users/user:id
Тіло запиту:
{
 "user": {
 "password": “newPass”,
 "id": user:id
 }
}
```

Цей запит призначений для зміни даних користувача з конкретним ідентифікатором user:id, внаслідок чого старий атрибут пароля повністю замінюється на нове значення newPass, і з цього моменту користувач із цим ідентифікатором матиме вже оновлений пароль. Останній етап життєвого циклу в концепції CDRUD відповідає за видалення об'єктів із системи, що реалізується за допомогою методу HTTP DELETE. Як приклад, якщо виникає необхідність повністю видалити профіль користувача з певним ідентифікатором user:id, програма звертається до бази даних і стирає запис, який точно відповідає цьому значенню. Для цього на сервер відправляється запит DELETE за адресою http://localhost:3399/users/user:id, після чого користувач із відповідним атрибутом id зникає з системи. Абсолютно аналогічні операції створення, читання, оновлення та видалення даних будуть спроектовані й реалізовані також і для моделі проекту.

2.2 Проектування та методи розробки компонентів системи

Платформа Angular базується на модульному методі розробки системи, де кожна частина додатка може функціонувати як окремий модуль, що забезпечує гнучке масштабування, розвиток та зміну програмного продукту. Будь-який проект у цій екосистемі містить щонайменше один корінний компонент, який об’єднує всю ієрархію інших компонентів з об’єктною моделлю документа DOM конкретної сторінки. Кожен такий компонент визначає клас із власними даними та внутрішньою логікою, а також пов'язує їх із макетом HTML для виведення

інформації на екран. Щоб вказати системі, що певний клас є саме компонентом, застосовується декоратор Component, під яким прописується шаблон та специфічні метадані.

Окрім компонентів, в Angular існують класи-сервіси, які позначаються декоратором Injectable. Вони не мають власного HTML-шаблону чи файлу зі стилями css, а їхні метадані відрізняються використанням властивості providedIn. Ця властивість визначає зону доступу до сервісу, і найпоширенішим її значенням є root, що дозволяє викликати та використовувати цей сервіс у будь-якому місці програми. Також у розробці активно застосовується декоратор Pipe, який дозволяє оголошувати функції для перетворення вхідних значень у потрібний формат безпосередньо перед їх відображенням у шаблоні. В Angular є стандартні інструменти такого типу, наприклад date, який трансформує формат дати. Якщо застосувати цей інструмент до змінної incomingDate із параметрами форматування дня, місяця та року, то на сторінці відобразиться рядок на кшталт дванадцяте березня 2022 року, незалежно від початкового вигляду даних у системі. За потреби користувачі можуть створювати власні унікальні класи такого типу.

Ще одним важливим елементом архітектури є директиви, які позначаються декоратором Directive. Шаблони Angular є динамічними, і під час їх відтворення фреймворк трансформує DOM відповідно до інструкцій, закладених у директивах. Технічно сам компонент також є директивою, проте через його центральну роль Angular виділяє для нього розширений декоратор Component, збагачений функціями для роботи з шаблонами. Усі директиви поділяють на структурні та атрибутивні. Структурні директиви безпосередньо змінюють макет сторінки, додаючи, видаляючи або замінюючи елементи в DOM для впровадження програмної логіки перед рендерингом. Типовими прикладами вбудованих структурних інструментів є ngIf, який працює як умовний оператор і вирішує доцільність виведення елемента на основі певної умови, а також ngFor, що функціонує як оператор циклу для послідовного відображення елементів масиву.

Саме директиву ngFor розробники використовують для того, щоб правильно вивести на екран весь список наявних проектів із коректними та актуальними даними для кожного з них.

```
<app-project-card *ngFor="let project of projects"
projectTitle="{{ project.title }}"
description="{{ project.description }}"
type="{{ project.type }}"
teamMembers="{{ project.workers.length }}"
creationDate="{{ project.creationDate }}">
</app-project-card>
```

Також приклад використання директиви *ngIf наведено нижче:

```
<button *ngIf="typeOfAction == 'login'"
class="action-button"
(click)="login()">
Log In
</button>

<button *ngIf="typeOfAction == 'registrastion'"
class="action-button"
(click)="createUser()">
Create Account
</button>
```

Однією з основних технологій односторінкових вебдодатків є маршрутизація, яка в екосистемі Angular реалізована за допомогою інструменту Angular Routing. Цей механізм функціонує таким чином, що замість виконання запиту до сервера для завантаження нової сторінки він просто динамічно показує або приховує певні частини інтерфейсу, які безпосередньо відповідають конкретним компонентам. Завдяки такому підходу користувач отримує миттєвий відгук від системи під час переходу між різними розділами, а загальна схема взаємодії та логіка роботи цієї технології в контексті додатка для управління проектами детально демонструє послідовність зміни станів програми без перезавантаження вкладки браузера.

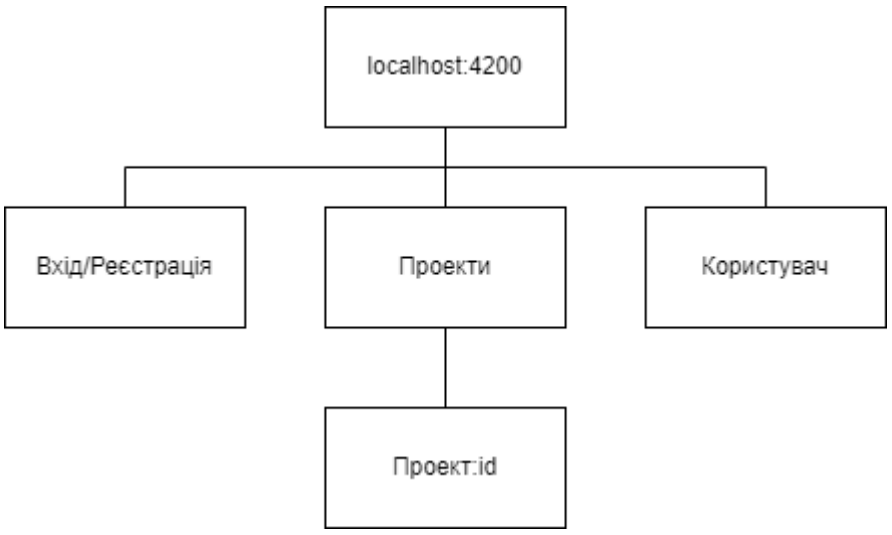


Рисунок 2.1 – Схема роботи маршрутизації

Однією з ключових переваг Angular Routing є технологія Lazy Loading, тобто відкладене або ліниве завантаження. За замовчуванням усі модулі NgModules завантажуються миттєво і одночасно під час старту програми, незалежно від того, чи є в них нагальна потреба для користувача в цей конкретний момент. Для масштабних додатків із розгалуженою системою маршрутів раціонально використовувати цей шаблон проектування, оскільки він забезпечує підвантаження необхідного модуля лише за безпосередньої вимоги. Завдяки відкладеному завантаженню суттєво зменшується початковий розмір пакетів даних, що безпосередньо впливає на скорочення часу першого запуску та підвищення швидкодії системи.

У свою чергу серверна частина цього програмного забезпечення функціонуватиме на базі технології Node.js із застосуванням фреймворку Express.js. Використання Express.js значно спрощує розробку серверної логіки вебдодатка, оскільки для створення та успішного запуску повноцінного сервера розробнику потрібно написати набагато менше рядків коду. Крім того, цей інструмент пропонує лаконічний та максимально зрозумілий синтаксис, що полегшує підтримку та подальше масштабування проекту, а базовий приклад створення такого сервера наочно демонструє мінімальну кількість налаштувань для обробки вхідних запитів.

```
const express = require('express');
const app = express();
app.get('/', (req, res) => res.send('Hello World!'));
app.listen(3000, () => console.log(`Example app listening on port 3000!`));
```

Для організації мережевих запитів у серверній частині використовуватиметься технологія Express Routes. Цей механізм необхідний для того, щоб під час надсилання запиту з клієнтської частини додатка сервер міг безпомилково розпізнати звернення, виконати правильні маніпуляції з даними та повернути очікуваний результат. Наприклад, рядок коду, який пов'язує адресу users із методом отримання всіх користувачів, демонструє базовий принцип обробки такого запиту через метод GET.

Після отримання запиту маршрутизатор перенаправляє його до конкретного методу, розташованого у файлі типу controller. Головне завдання контролера полягає в первинній обробці вхідних даних, перевірці наявності всіх необхідних параметрів і негайному поверненні помилки у разі виявлення будь-яких невідповідностей. Якщо перевірка проходить успішно, контролер викликає метод із файлу типу service. Саме у сервісному шарі зосереджена вся основна бізнес-логіка програми, відбуваються складні обчислення та здійснюються безпосередні запити до бази даних для вилучення чи зміни інформації, що дозволяє чітко розмежувати обов'язки між різними рівнями архітектури вебдодатка.

2.3 Проектування інтерфейсу додатку

Найпростішим типом вебкерування є веб-інтерфейс користувача або веб-інтерфейс. Його функціонування безпосередньо пов’язане з можливостями взаємодії за принципом наведи та вибери. Такий підхід дозволяє людині швидко та ефективно керувати додатком за допомогою базових пристроїв введення даних, якими є клавіатура та мишка.

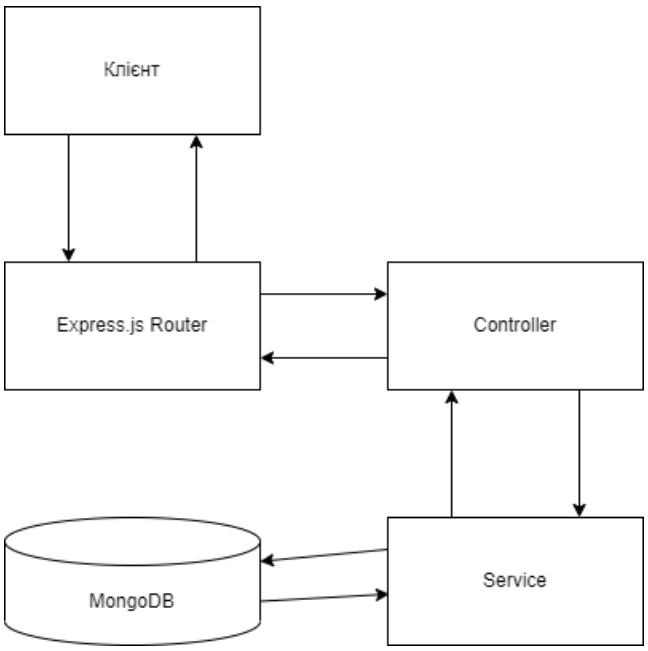


Рисунок 2.2 – Схема взаємодії всіх частин системи

Взаємодія з будь-яким сучасним вебдодатком традиційно базується на двох ключових аспектів, якими є користувацький інтерфейс UI та користувацький досвід UX. Процес проектування UI зосереджений перш за все на зовнішньому вигляді, естетиці та стилі програмного забезпечення або цифрових пристроїв. Головна мета дизайнерів у цьому напрямку полягає у створенні таких графічних або навіть голосових інтерфейсів, які будуть максимально приємними для сприйняття та простими у використанні. У свою чергу проектування UX охоплює набагато ширший спектр завдань і спрямоване на формування значущого та релевантного досвіду для кінцевого споживача. Цей підхід включає розробку та інтеграцію всього життєвого циклу продукту, починаючи від аспектів брендингу та дизайну і закінчуючи загальною зручністю використання та аналізом функціональності платформи.

Для безпосереднього технічного створення візуальної частини будь-якого вебдодатка застосовується класичне тріо базових технологій, де HTML виступає як мова гіпертекстової розмітки. Вона відповідає за побудову структури та каркаса сторінки за допомогою різних тегів, серед яких універсальні блоки div,

гіперпосилання a, текстові абзаци p, зображення img, шапки сайтів header, а також інтерактивні форми form з полями введення input.

Наступна технологія CSS, відома як каскадні таблиці стилів, відповідає за зовнішнє оформлення цієї розмітки, дозволяючи задавати кольорову гаму, текстові шрифти та просторове розміщення елементів. Для точного вибору потрібного об'єкта в стилях застосовують спеціальні селектори, якими можуть виступати класи, унікальні ідентифікатори або безпосередньо назви самих елементів розмітки.

Кінцевим елементом є мова програмування JavaScript, яка використовується для динамічного пожвавлення інтерфейсу та забезпечення його інтерактивності. Завдяки її можливостям додаток здатний миттєво реагувати на дії користувача, відправляти асинхронні запити на сервер для отримання свіжих даних або змінювати колірну тему сайту. Реалізація такої поведінки відбувається за допомогою спеціальних слухачів подій Event Listener, які прив'язуються до конкретних маніпуляцій, наприклад до кліку мишки або наведення курсора, та автоматично запускають відповідну логічну функцію після виконання цієї дії.

Всі компоненти користувацького веб-інтерфейсу для системи управління проектами були детально спроектовані та розроблені в сучасному графічному редакторі Figma. Візуальне оформлення первинної сторінки, яка відповідає за вхід користувача до системи та його автентифікацію, наочно продемонстровано на рис. 2.3. У свою чергу загальний вигляд та структура головної сторінки додатка, де відображається весь доступний список поточних проектів у вигляді інформаційних карток, представлені на рис. 2.4.

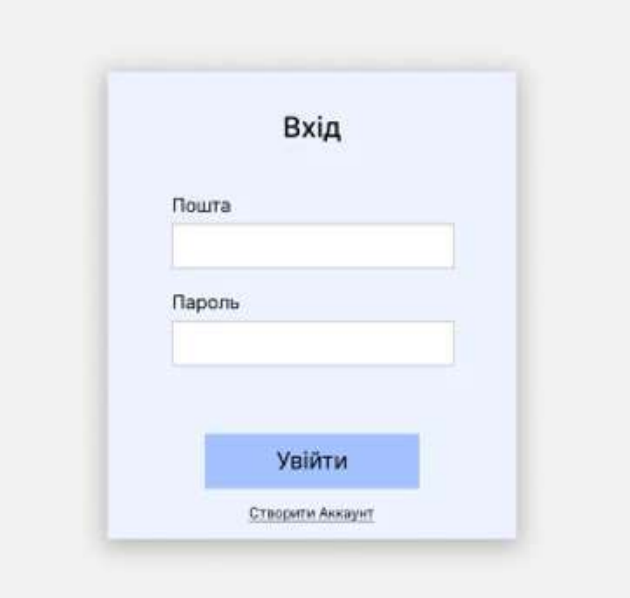


Рисунок 2.3 – Сторінка входу в систему

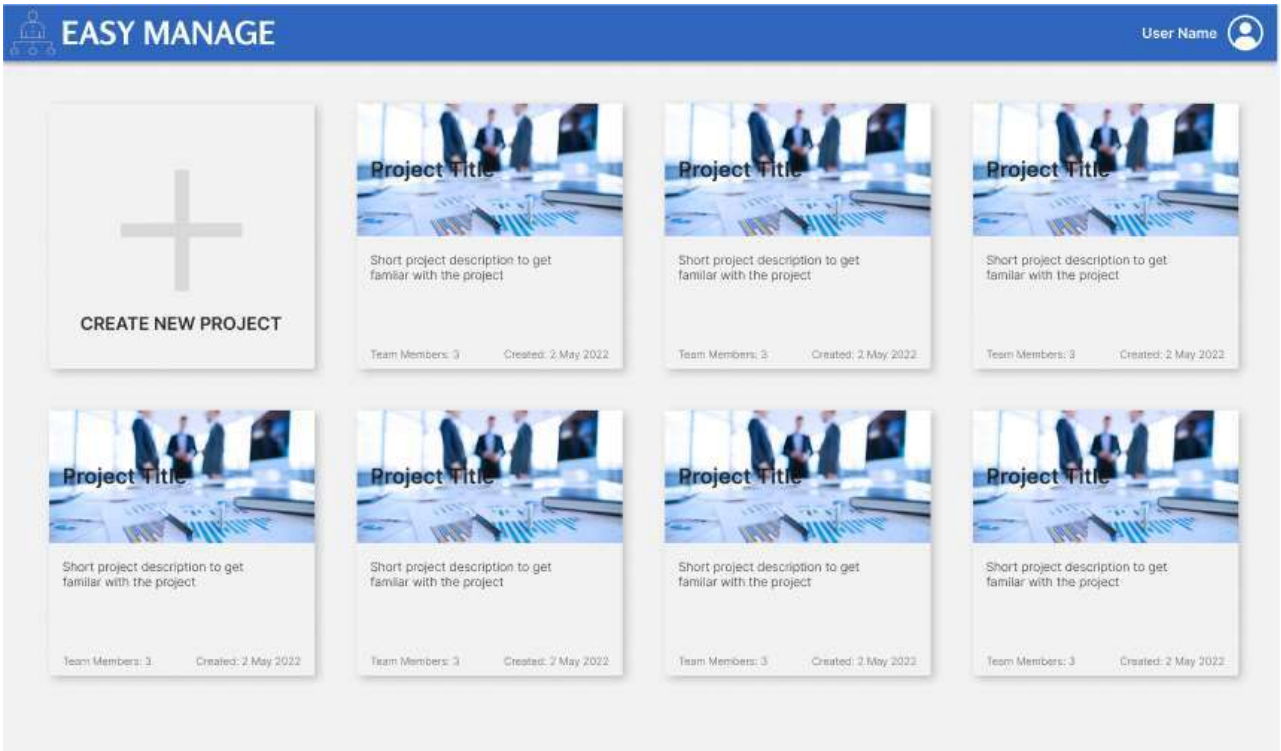


Рисунок 2.4 – Головна сторінка зі списком проектів

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ПРОЕКТАМИ

3.1 Підготовка середовища для розробки

Процес розробки цього додатка відбуватиметься в популярному редакторі коду Visual Studio Code, відомому як VS Code. Цей безкоштовний інструмент суттєво полегшує роботу розробника програмного забезпечення, допомагаючи ефективно писати, налагоджувати та автоматично виправляти код за допомогою інтелектуальної технології intelli-sense. Важливою перевагою цього редактора є його глибока інтеграція із системою контролю версій Git, завдяки чому можна легко фіксувати зміни, керувати історією розробки та за потреби швидко повертатися до попередніх контрольних точок. Крім того, VS Code підтримує величезну кількість плагінів і спеціальних розширень для оптимізації робочого процесу. Серед них можна виділити ES Lint, який відповідає за гарне структурування та упорядкування коду відповідно до стандартів, Bracket Pair для зручної орієнтації в ієрархії відкритих дужок і різноманітні кольорові теми, що дозволяють повністю кастомізувати інтерфейс під власні вподобання.

Робочий простір самого редактора VS Code гнучко розподіляється на кілька функціональних зон. Мінімальний варіант передбачає наявність лише одного вікна з вмістом поточного файлу, проте найзручнішим форматом є робота з двома зонами. У такому режимі зліва відкривається бічна панель провідника, де можна шукати та відкривати інші файли як у межах поточного проекту, так і за його межами. Цю ж ліву панель можна використовувати для перемикання між меню встановлених розширень, інструментами контролю версій та модулем дебагу коду, а загальний вигляд цього інтерфейсу детально продемонстровано на рис. 3.1.

Для безпосереднього відстеження та контролю версій коду в проекті застосовується технологія Git, тоді як для віддаленого хостингу, спільної роботи та надійного збереження всієї кодової бази в мережі використовуватиметься вебсервіс

BitBucket. Ця хмарна платформа надає розробникам усі необхідні інструменти для безпечного керування репозиторіями, а також дозволяє зручно фіксувати, переглядати та контролювати будь-які поточні зміни в коді.

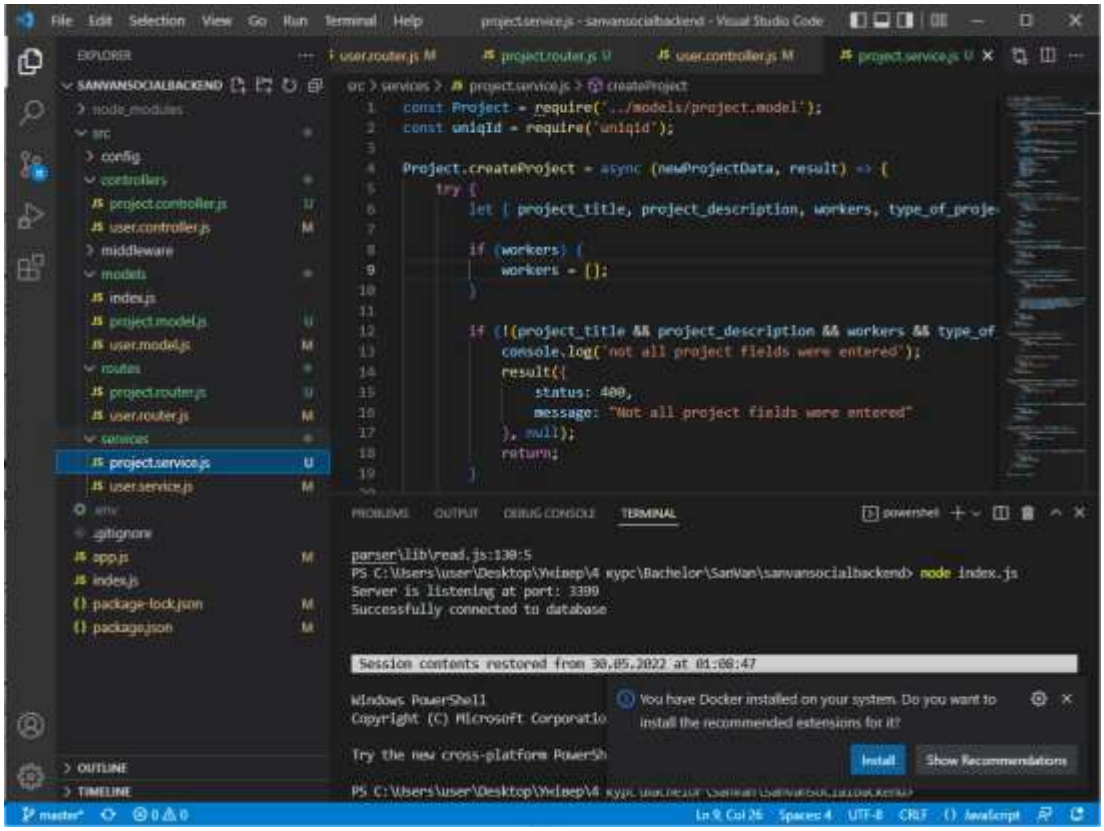


Рисунок 3.1 – Інтерфейс редактора коду VS Code

Хмарний сервіс BitBucket надає зручну платформу для розміщення віддалених репозиторіїв Git. Його візуальний інтерфейс є досить простим та зрозумілим навіть для розробників-початківців, що дозволяє їм легко використовувати всі переваги контролю версій. Це суттєво спрощує процес, оскільки робота з чистим Git зазвичай вимагає глибших технічних знань та постійної взаємодії з командним рядком.

Головна відмінність Git від багатьох інших систем, як-от Subversion, полягає в принципово іншому підході до сприйняття та обробки даних. Більшість традиційних систем контролю версій зберігають інформацію у вигляді послідовного списку файлових редагувань, тобто вони розглядають дані як

сукупність базових файлів і фіксацію змін, що відбувалися з кожним із них протягом певного часу.

Натомість технологія Git не обробляє і не накопичує інформацію подібним чином. Вона сприймає поточні дані радше як набір знімків усієї файлової системи. Щоразу, коли розробник створює новий коміт і фіксує поточний стан проекту, Git повністю запам’ятовує вигляд усіх файлів у цей конкретний момент часу та зберігає пряме посилання на цей знімок. Для досягнення максимальної ефективності та економії дискового простору система не дублює ті файли, які не зазнали жодних змін з моменту попередньої фіксації, а просто створює посилання на вже наявний ідентичний файл, що детально відображено на архітектурній схемі зберігання даних на рис. 3.2.

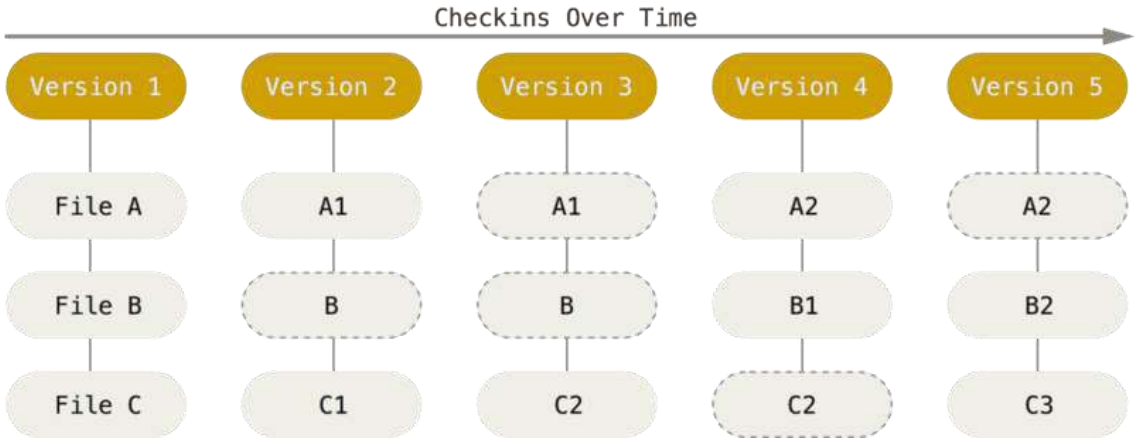


Рисунок 3.2 – Схема роботи зберігання даних Git

Для ініціалізації клієнтської частини проекту в системі Git використовувались такі команди:

– git init;

– git add .;

– git clone IvanKryvenko/SanVanFront.git;

– git commit -m "first commit";

– git push -u origin master.

Подібні команди застосовувалися і для ініціалізації серверної частини

додатка, оскільки ці два репозиторії повністю ізольовані та функціонують окремо один від одного. Безпосередньо для створення та первинного налаштування клієнтської частини проекту використовувалися інструменти командного рядка, де спочатку за допомогою менеджера пакетів виконувалося глобальне встановлення останньої версії інтерфейсу командного рядка Angular CLI на комп'ютер через команду npm install -g @angular/cli. Після цього запускалася команда ng new san-van-front, яка автоматично генерувала структуру та ініціалізувала новий проект. У результаті подальшого доопрацювання та фінального структурування репозиторію було сформовано кінцеву архітектуру каталогів, загальний вигляд якої детально продемонстровано на рис. 3.3.

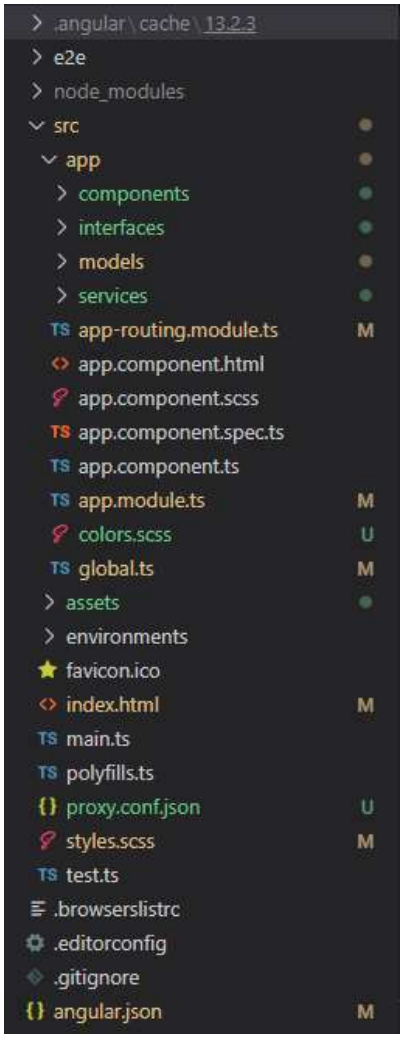


Рисунок 3.3 – Структура клієнтської частини проекту

Директорія e2e повністю відповідає за зберігання та виконання наскрізних тестів додатка, тоді як каталог node_modules містить усі допоміжні бібліотеки, сторонні залежності та інструменти, необхідні для комфортної розробки. Основні вихідні файли проекту зосереджені в папці src, де розміщуються корінний компонент app.component, а також головні файли маршрутизації та модулів усієї системи. Всередині базового каталогу app функціонує чітко розмежована структура підпапок, де директорія components містить окремі перевикористовувані компоненти інтерфейсу, наприклад кнопки, інформаційні картки проектів або верхнє навігаційне меню header. Папка interfaces призначена для зберігання інтерфейсів, які описують типи полів та структуру моделей об’єктів системи, зокрема для сутностей user та project. У свою чергу каталог models акумулює в собі великі функціональні модулі сторінок додатка, такі як home page, user page та login page, а директорія services містить усі необхідні класи служб для забезпечення логіки клієнтської частини. Для створення серверної частини проекту розробнику достатньо сформувати власну необхідну структуру файлів, загальний вигляд якої детально представлений на рис. 3.4.

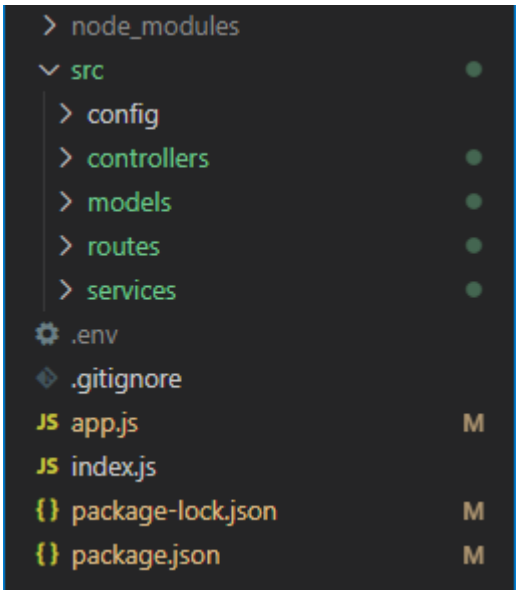


Рисунок 3.4 – Структура серверної частини проекту

На серверній частині додатка директорія node_modules виконує ту саму

функцію, що й на клієнтській стороні, зберігаючи всі необхідні сторонні залежності. При цьому файли app.js та index.js відповідають за первинну конфігурацію сервера, збирання коду всього проекту та його безпосередній запуск. Уся внутрішня логіка зосереджена в каталозі src, де папка config містить файли налаштування для підключення до бази даних, а в папці controllers розміщуються файли первинної обробки та перевірки вхідних параметрів перед їх передачею в сервісний шар. Директорія models слугує для опису структур моделей користувача та проекту, які безпосередньо взаємодіють із базою даних, тоді як у папці routes описується вся маршрутизація для кінцевих точок, до яких клієнт може звертатися по API. Кінцевим елементом цієї структури є каталог services, що містить файли для виконання основних логічних функцій та глибокої обробки інформації.

Для встановлення та керування всіма додатковими допоміжними бібліотеками чи пакетами в проекті застосовується менеджер пакетів npm. Цей інструмент є найбільшим у світі реєстром програмного забезпечення, який налічує понад 800 000 пакетів готового коду. Завдяки такій екосистемі розробники з відкритим кодом по всьому світу можуть легко ділитися своїми програмними рішеннями та інтегрувати готові модулі у власні розробки для прискорення процесу створення продуктів.

3.2 Розробка компонентів для обробки запитів та інтерфейсу

Для забезпечення правильної обробки, оновлення та отримання всієї необхідної інформації розробляється спеціальний інтерфейс прикладного програмування API, який дозволяє клієнтській частині додатка ефективно взаємодіяти із серверною стороною на основі чіткого набору визначень і протоколів. У рамках створеної системи управління проектами було спроектовано ряд кінцевих точок для виконання конкретних операцій. Так, реєстрація нового користувача та його подальша авторизація в системі реалізуються через запити

POST на адреси register та login відповідно. Отримання повного списку зареєстрованих у програмі людей відбувається за допомогою методу GET на кінцеву точку users.

Робота з сутністю проектів охоплює ширший спектр інтеракцій. Створення нового проекту виконується через запит POST на адресу createProject, тоді як для повного редагування загальної інформації про проект застосовується метод PUT на кінцеву точку editProject. Отримання відфільтрованого списку проектів, закріплених за конкретним користувачем, а також вилучення детальних даних про один вибраний проект забезпечуються методами GET через кінцеві точки getProjectForUser та projectDetail. Повне видалення проекту з бази даних реалізовано за допомогою методу DELETE на адресу projectDelete. Крім того, динамічне оновлення внутрішнього складу учасників або редагування проектних завдань разом із актуалізацією їхніх поточних статусів покладено на методи PUT, які надсилають запити на кінцеві точки projectUpdateMembers та projectUpdateTasks.

Основою для передачі всієї цієї інформації в мережі інтернет виступає протокол прикладного рівня HTTP, призначений для розподілених і спільних гіпермедійних систем. Цей протокол функціонує без збереження стану, що робить його універсальним інструментом для різноманітних мережевих цілей, а його можливості гнучко розширюються завдяки використанню специфічних методів запиту, заголовків та кодів відповідей сервера. Загальні засади взаємодії між клієнтом і сервером у рамках цього протоколу, включаючи цикли надсилання запитів та отримання відповідей, детально відображені в базовій архітектурі роботи мережі на рис. 3.5.

Мережевий протокол HTTP за своєю суттю є протоколом запиту та відповіді, який повністю базується на класичній архітектурі клієнт-сервер. У цій схемі взаємодії різноманітні веб-браузери, автоматизовані роботи та пошукові системи виступають у ролі клієнтів, тоді як веб-сервер виконує функції серверної сторони,

яка обробляє надходження та видачу інформації.

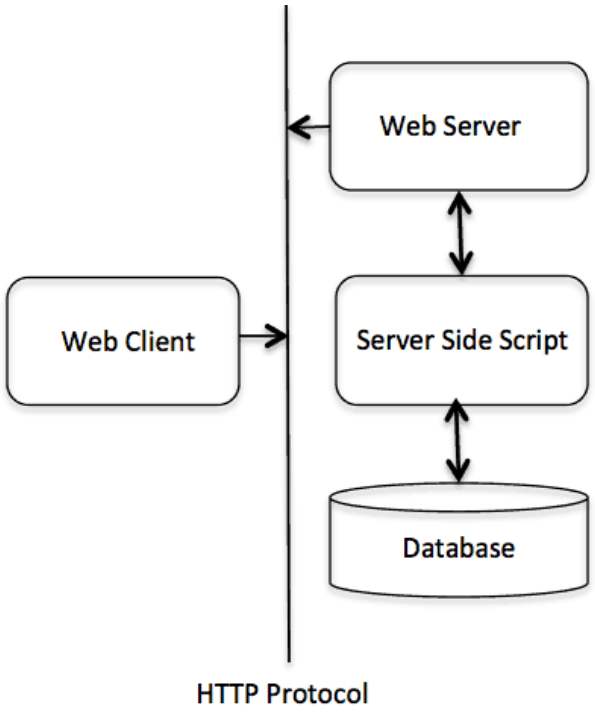


Рисунок 3.5 – Базова архітектура роботи HTTP протоколу

З метою суттєвого підвищення безпеки та надійного шифрування даних у майбутньому можна буде запровадити прогресивніший протокол HTTPS. Ця технологія розширює можливості стандартного HTTP завдяки додаванню захищеного шару, що дозволяє ефективно оберігати цілісність і конфіденційність усієї інформації, яка передається між персональним комп'ютером користувача та віддаленим сайтом. На поточному етапі створення та тестування цього вебдодатка використання захищеної версії протоколу не є критично необхідним або доцільним, оскільки програма ще не розміщена на постійному хостингу. Проте під час подальшого розширення загального функціоналу, масштабування проекту та його повноцінного запуску в глобальній мережі таке безпекове впровадження стане повністю виправданим.

Усі ці мережеві запити відбуваються за певними адресами, а базову структуру уніфікованого покажчика URL, який визначає шлях до кожного ресурсу в інтернет-

просторі, наочно продемонстровано на рис. 3.6.



Рисунок 3.6 – Базова структура URL

Прикладом відправки запиту з клієнтської сторони додатка може слугувати наступний фрагмент програмного коду, який призначений для забезпечення процедури реєстрації нового користувача в системі шляхом передачі його персональних даних на сервер.

```
createNewUser(newUserDetails): Observable<any> {
 let headers = new HttpHeaders();
let params = new HttpParams();

params = params.append('firstName', newUserDetails.firstName); params =
params.append('lastName', newUserDetails.lastName); params = params.append('email',
newUserDetails.email);
params = params.append('password', newUserDetails.password);

return this.http.post(`${API_URL}/register`, params, {headers});

}
```

Змінна API_URL містить у собі значення базового посилання, на яке безпосередньо надходитиме мережевий запит, і в цьому конкретному випадку вона вказує на адресу http://localhost:4200, оскільки саме там запущено та функціонує сервер. Візуальний інтерфейс для цього вебдодатка побудований переважно на базі мови гіпертекстової розмітки HTML та оформлений за допомогою каскадних таблиць стилів CSS із залученням сучасного препроцесора SCSS. Використання

SCSS дозволяє істотно спростити та прискорити написання стилістичних правил завдяки впровадженню спеціальних домішок mixin, які працюють як багаторазові функції або готові фрагменти коду, що їх можна легко підключати в будь-яких місцях проекту.

З метою оптимізації робочого часу та спрощення загального процесу створення інтерфейсу під час розробки були активно задіяні сторонні бібліотеки готових елементів. Однією з них стала Angular Material UI, яка є набором нативних і повністю адаптованих компонентів, створених спеціально для додатків на Angular. Цей інструмент вирізняється високою продуктивністю та легкістю в інтеграції, а для потреб цього проекту з нього були взяті такі модулі, як MatDialogModule для модальних вікон, MatInputModule для текстових полів введення, MatButtonModule для інтерактивних кнопок та MatSelectModule для випадних списків. Іншою важливою бібліотекою готових рішень стала PrimeNG, яка пропонує величезну колекцію з понад вісімдесяти різноманітних елементів, тем і готових шаблонів. З цієї платформи розробники запозичили спеціалізований компонент DragAndDrop, який було успішно інтегровано в інтерактивну дошку проекту для забезпечення зручного візуального перетягування карток та швидкої зміни поточного статусу завдань.

Внутрішня структура самих компонентів у фреймворку Angular є максимально простою, логічною та зрозумілою для розробника, оскільки кожен окремий елемент інтерфейсу зазвичай розділяється на три або чотири автономні файли, які відповідають за функціональну логіку, HTML-шаблон, зовнішні стилі та, у деяких випадках, за автоматизовані тести. Безпосередньо функціональна частина компонента створюється мовою програмування TypeScript. Саме у цьому файлі прописується первинна ініціалізація даних, здійснюється імпорт усіх необхідних допоміжних модулів чи сторонніх бібліотек, а також оголошуються змінні та прописуються ключові метадані, а базовий приклад реалізації такого файлу для компонента проекту наочно ілюструє початкове налаштування його логіки.

```
import { ProjectService } from 'src/app/services/project.service';
import { Router } from '@angular/router';
@Component({
selector: 'app-projects-list',
templateUrl: './projects-list.component.html',
styleUrls: ['./projects-list.component.scss']
})
export class ProjectsListComponent implements OnInit {}
```

Файл із HTML-шаблоном не є критично обов'язковим елементом для кожного об'єкта, оскільки структуру розмітки за бажання можна прописати безпосередньо всередині функціонального файлу TypeScript у розділі метаданих, а також він повністю відсутній у директивах, які не виконують роль компонентів. Проте для забезпечення максимальної зручності під час розробки та полегшення розуміння коду зазвичай створюють окремий файл шаблону, який за своєю суттю є звичайним HTML-документом, але містить специфічні особливості та директиви, притаманні виключно фреймворку Angular.

Аналогічно і файл із зовнішніми стилями може підтримувати різні формати залежно від обраного препроцесора, причому в рамках цього конкретного проекту розробники зупинили свій вибір на технології SCSS. Цей файл також належить до категорії необов'язкових елементів архітектури, оскільки всі стилістичні правила можна за потреби декларувати безпосередньо в метаданих самого функціонального класу.

У свою чергу файл з автоматизованими тестами для компонента теж не є обов'язковим для успішного функціонування програми, і на відміну від розмітки чи стилів його неможливо інтегрувати всередину метаданих основного класу. Головна його функція полягає виключно у проведенні автоматичного тестування працездатності ізольованого елемента, а стандартна структура організації всіх цих файлів у межах створеного вебдодатка має чітко структурований і послідовний вигляд.

home.component.ts

home.component.html

home.component.scss

home.component.spec.ts

Особливостями Angular шаблонів, про які згадувалось вище, на відміну від стандартних HTML файлів є директорії, трохи інший запис подій слухачів подій, запис властивостей та інтерполяція – спосіб відображення інформації, що зберігається в змінних або отримується шляхом виклику методів компоненту всередині функціонального файлу, використовує подвійні фігурні дужки {{}}.

Основними директоріями, які використовуються в проекті є структурні директорії, які були описані в розділі 2.2, а саме *ngIf та *ngFor. *ngFor використовується для відображення списку проектів, що доступні для користувача:

```
<app-project-card *ngFor="let project of projects"
(click)="openProject(project.id)"
projectTitle="{{ project.project_title }}"
description="{{ project.project_description }}"
type="{{ project.type_of_project }}"
teamMembers="{{ project.workers.length }}"
creationDate="{{ project.creation_date }}">
</app-project-card>
```

*ngIf використовувалась для перевірки наявності певних змінних, які необхідні для рендерингу:

```
<div *ngIf="detailsLoaded | async" class="project-board">
<div *ngFor="let step of projectDetail.steps" class="step">
{{ step.replaceAll('_', ' ') }}
</div>
</div>
```

Повний приклад структури всього компоненту на прикладі компоненту зі списком проектів project-list.component наведено нижче:

project-list.component.ts

```
import { Component, OnInit } from '@angular/core';
import { MatDialog, MatDialogConfig } from "@angular/material/dialog";
import { CreateProjectModalComponent } from '../create-project-modal/create- project-
modal.component';
import { ProjectService } from 'src/app/services/project.service';
import { Router } from '@angular/router';
@Component({
selector: 'app-projects-list',
templateUrl: './projects-list.component.html',
styleUrls: ['./projects-list.component.scss']
})
export class ProjectsListComponent implements OnInit {
isModalVisible = false;
projects = [];
constructor(
private modal: MatDialog,
public projectService: ProjectService,
public router: Router
) { }
ngOnInit(): void {
this.getProjectsList();
}
getProjectsList(): void {
this.projectService.getProjects().subscribe(data => {
this.projects = JSON.parse(data.data).projectsList;
}, (err) => {
console.error(err);
});
}
openModal() {
const modalConfig = new MatDialogConfig();
modalConfig.autoFocus = true;
modalConfig.hasBackdrop = true;
modalConfig.width = '600px';
let modalRef = this.modal.open(CreateProjectModalComponent, modalConfig);
modalRef.afterClosed().subscribe(result => {
this.getProjectsList();
});
}
openProject(projectId: string): any {
this.router.navigate(['/project', projectId]);
} }
```

project-list.component.html

```
<div class="projects-list">
 <div class="projects-list-container">
 <div class="create-project" (click)="openModal()">
 <img class="add-icon" src="../../../assets/images/add.svg"> <h2>Створити проект</h2>
 </div>
 <app-project-card *ngFor="let project of projects"
 (click)="openProject(project.id)"
 projectTitle="{{ project.project_title }}"
 description="{{ project.project_description }}"
 type="{{ project.type_of_project }}"
 teamMembers="{{ project.workers.length }}"
 creationDate="{{ project.creation_date }}">
 </app-project-card>
</div>
</div>
```

project-list.component.scss

```
@import '../../colors.scss';
.projects-list {
height: 100%;
padding: 3rem;

.projects-list-container {
display: flex;
flex-wrap: wrap;
gap: 3rem;

.create-project {
height: 300px;
width: 300px;
@include glass-background;
display: flex;
flex-direction: column;
align-items: center;
justify-content: center;
color: $grey;
@include shadow;
transition: 0.3s;
cursor: pointer;
 .add-icon {
height: 150px;
}
&:hover {
transform: translateY(-5px);
}}}}
```

Внаслідок фінального об’єднання всіх структурних частин компонента в єдине ціле, а також його подальшої інтеграції та взаємодії з іншими автономними елементами системи розробники отримали готовий користувацький інтерфейс, візуальний вигляд якого детально зображений на рис. 3.7.

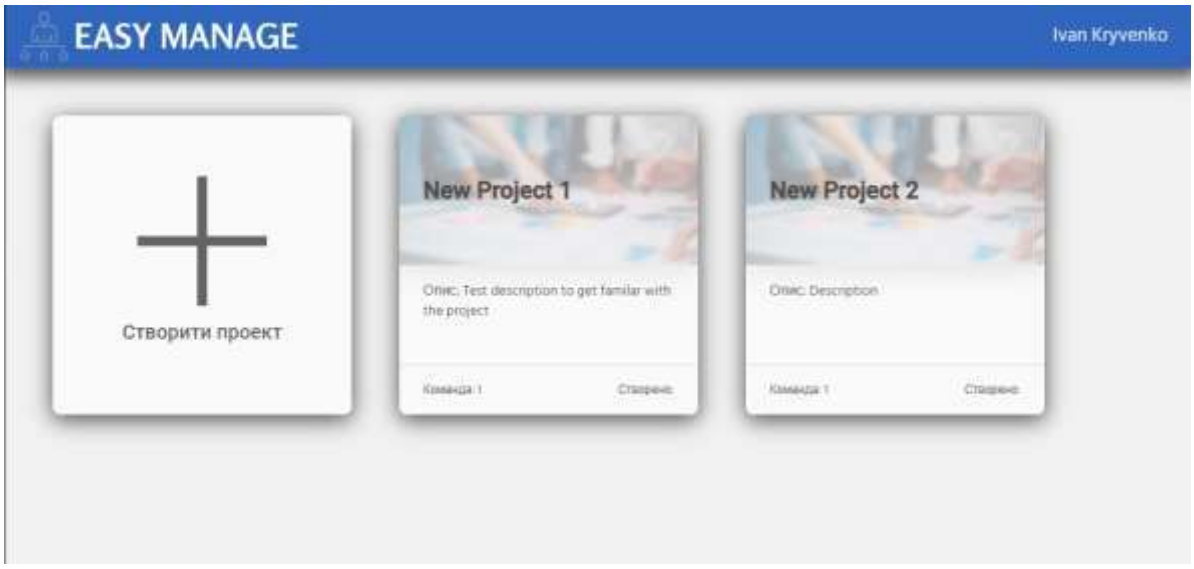


Рисунок 3.7 – Інтерфейс списку проектів, доступних користувачу

Спроектований користувацький інтерфейс повноцінно підтримує весь спектр базових функцій, які є критично необхідними для сучасної системи управління проектами. Візуальне оформлення та логіка роботи ізольованого компонента, який відповідає за автентифікацію та вхід в наявний обліковий запис, детально продемонстровані на рис. 3.8, тоді як інтерфейсна форма для реєстрації нового користувача в системі представлена на рис. 3.9.

Головна сторінка, на якій відображається весь перелік доступних користувачу проектів, показана на рис. 3.7. Загалом загальна кількість зареєстрованих проектів у базі даних системи може бути надзвичайно великою, проте діє суворе розмежування прав доступу, завдяки якому конкретна людина бачить у своєму робочому просторі виключно ті проекти, які були створені безпосередньо нею, або ті, де вона офіційно фігурує як один із запрошених учасників команди.



Рисунок 3.8 – Вхід в обліковий запис

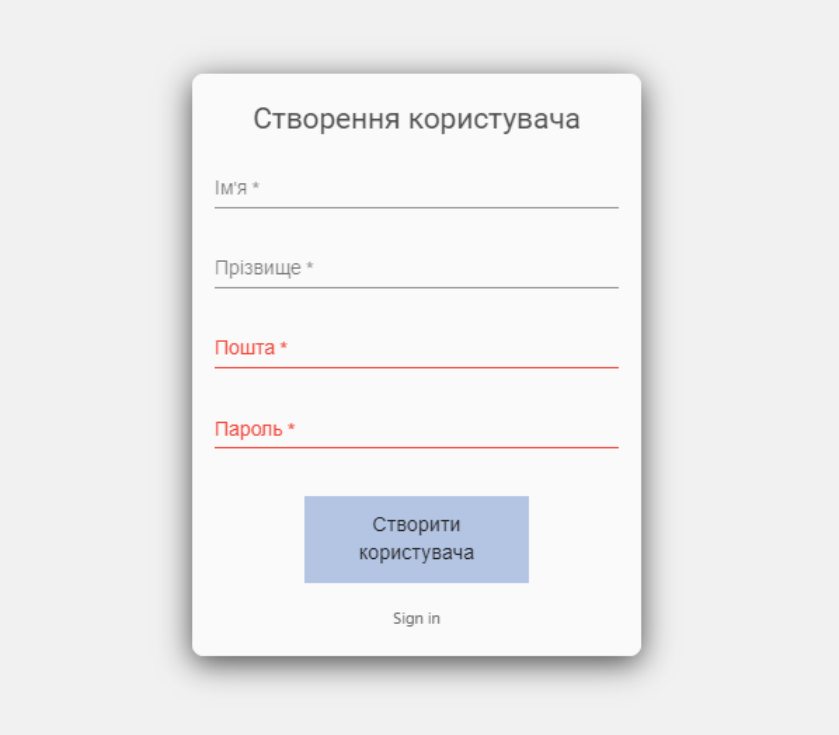


Рисунок 3.9 – Створення нового користувача

Модальне вікно для створення нового проекту зображене на рис. 3.10. Сторінка та дошка проекту зображена на рис. 3.11.

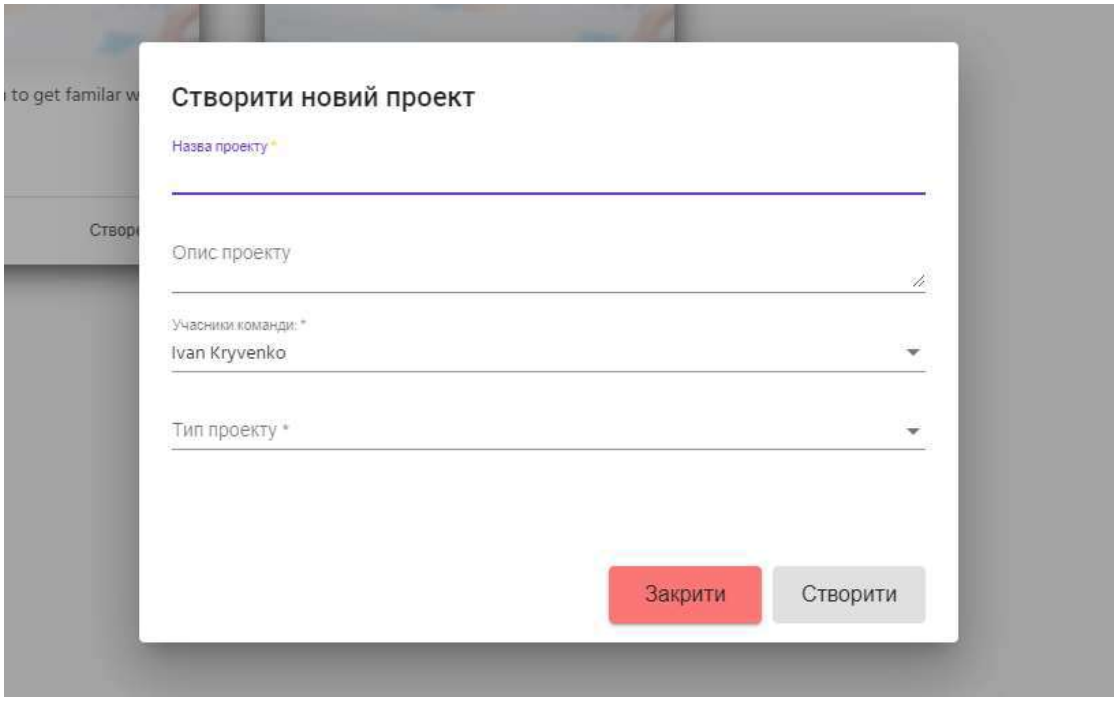


Рисунок 3.10 – Модальне вікно створення проекту



Рисунок 3.11 – Сторінка та дошка нового проекту

3.3 Тестування програмного забезпечення

Процес перевірки працездатності створеної системи здійснюватиметься на основі комплексу чітких функціональних критеріїв, які охоплюють усі етапи взаємодії користувача з платформою. Згідно з визначеною логікою, під час первинного відкриття вебдодатка система автоматично перевіряє стан

автентифікації, і в разі відсутності активної сесії користувача негайно перенаправляє на сторінку авторизації та реєстрації. На цьому етапі спрацьовують захисні алгоритми, тому спроба надіслати порожню форму без введення електронної пошти чи пароля призводить до миттєвого виведення повідомлення про помилку. Важливим технічним маркером є те, що до моменту успішного входу в локальному сховищі браузера local storage повністю відсутня будь-яка конфіденційна інформація про поточного користувача. Крім того, для щойно зареєстрованих осіб, які ще не мають створених репозиторіїв, робочий простір головного екрана залишатиметься абсолютно порожнім. Надалі діє суворе розмежування прав доступу, завдяки якому кожна людина бачить у своєму профілі виключно ті проекти, які вона розробила особисто або до яких її долучили як офіційного учасника команди.

Подальша навігація всередині платформи передбачає, що клік по інформаційній картці конкретного проекту автоматично переводить розробника на цільову сторінку з інтерактивною дошкою завдань. Функціонал цієї дошки дозволяє безперешкодно генерувати нові завдання в межах будь-якої доступної колонки статусів. Сама зміна поточного статусу обраного завдання реалізована максимально інтуїтивно через візуальне переміщення картки з одного стовпця в інший за допомогою звичайного курсору мишки. Користувачі також мають повний набір прав для редагування внутрішнього змісту або остаточного видалення будь-яких завдань. Безпека персональних даних забезпечується тим, що під час виходу з облікового запису або штатного завершення сесії роботи всі тимчасові записи та токени, які зберігалися в local storage, безповоротно знищуються.

Практична реалізація та тестування системи за цими критеріями повністю підтвердили закладену логіку. Зокрема, під час запуску програми неавторизований відвідувач дійсно бачить вікно входу, представлене на рис. 3.12, а некоректне заповнення полів викликає відповідне попередження, показане на рис. 3.13. У разі введення правильних ідентифікаційних даних відбувається успішний перехід на

головну сторінку з переліком проектів, як це продемонстровано на рис. 3.14, а в локальну пам'ять браузера local storage автоматично записуються службові відомості та унікальний токен авторизації, що наочно зафіксовано на рис. 3.15. Для розширення робочого простору та створення нового проекту користувачеві достатньо натиснути на спеціальну картку зі знаком плюс, що активує появу зручного модального вікна для введення первинних метаданих та параметрів майбутнього проекту, як це детально зображено на рис. 3.16.



Рисунок 3.12 – Сторінка авторизації

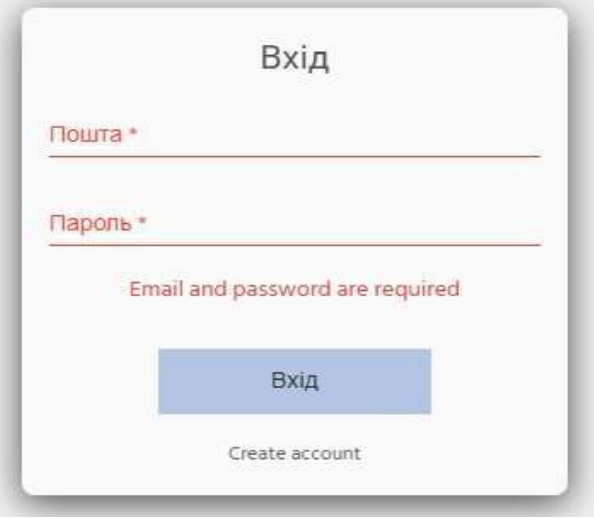


Рисунок 3.13 – Помилка при авторизації

Під час переходу на цільову сторінку конкретного проекту, яка детально зображена на рис. 3.17, користувачеві відкривається інтерактивна робоча дошка. На цьому просторі розміщені окремі функціональні колонки із картками завдань, причому кожен такий стовпець безпосередньо відображає поточний статус виконання того чи іншого завдання.

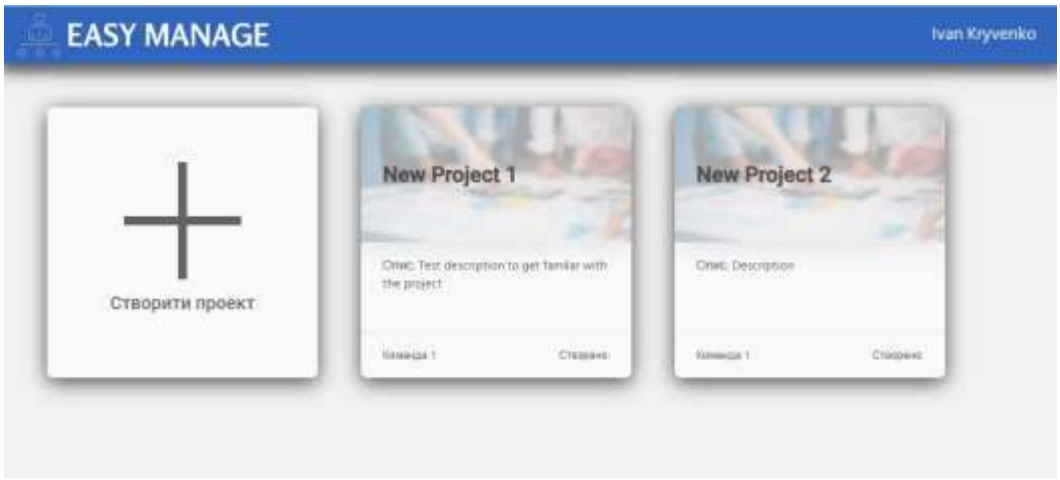


Рисунок 3.14 – Головна сторінка зі списком проектів

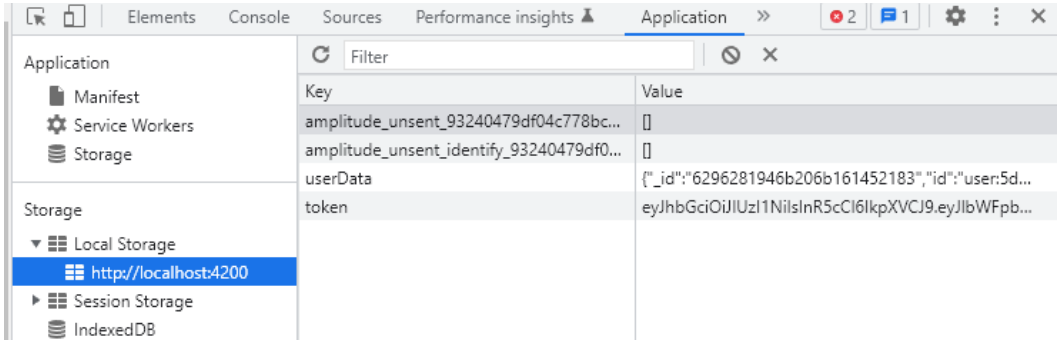


Рисунок 3.15 – Дані користувача в local storage

Для створення нового завдання користувачеві достатньо натиснути на спеціальну кнопку зі знаком плюс, що призводить до миттєвої появи безпосередньо всередині обраної колонки зручної форми з текстовими полями для детального опису запланованої роботи та функціональним меню вибору конкретного виконавця, який відповідатиме за цей шмат роботи, як це наочно продемонстровано

на рис. 3.18. Гнучке керування робочим процесом також дозволяє оперативно змінювати поточний статус будь-якого завдання шляхом його простого візуального перетягування між стовпцями за допомогою звичайного курсору мишки, що детально зафіксовано на рис. 3.19. Крім того, керівник або учасник команди має можливість у будь-який момент здійснити комплексний моніторинг стану виконання всіх поточних завдань та оцінити прогрес проекту в цілому. Ця функція реалізована через спеціальне модальне вікно аналітики та статистики, яке викликається кліком на кнопку із зображенням діаграми у правій частині загальної панелі управління, як показано на рис. 3.20.

Після успішного виконання всіх запланованих дій та штатного завершення поточної робочої сесії користувач натискає на кнопку виходу. У цей момент система безпеки автоматично перенаправляє його назад на стартову сторінку авторизації, паралельно повністю очищуючи локальне сховище браузера local storage від будь-яких персональних даних, службових записів та активних токенів ідентифікації, що детально відображено на рис. 3.21.

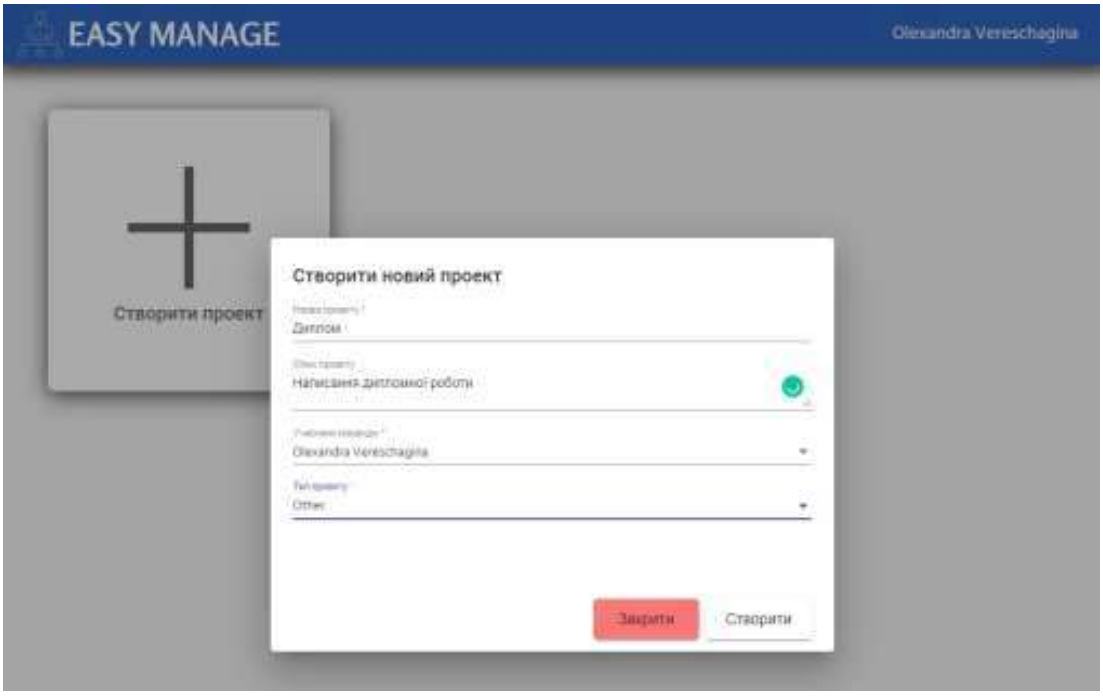


Рисунок 3.16 – Модальне вікно створення нового проекту



Рисунок 3.17 – Сторінка проекту

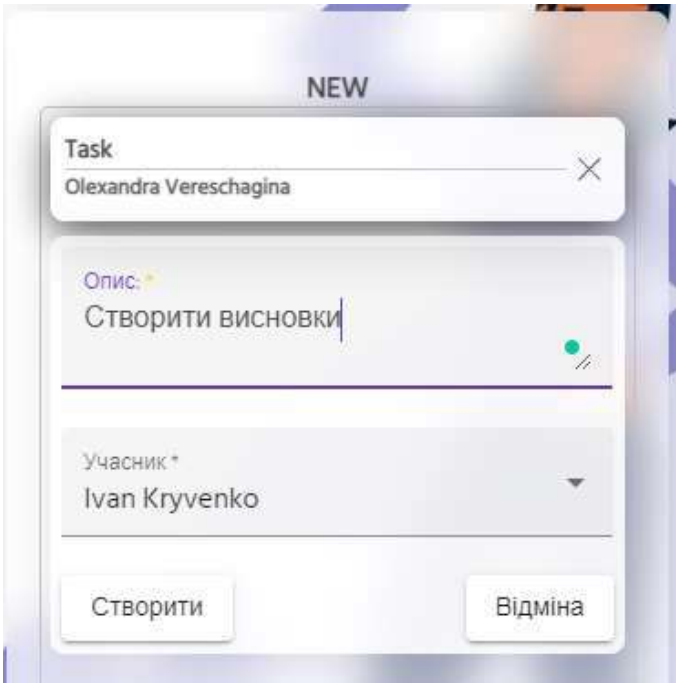


Рисунок 3.18 – Створення завдання для проекту

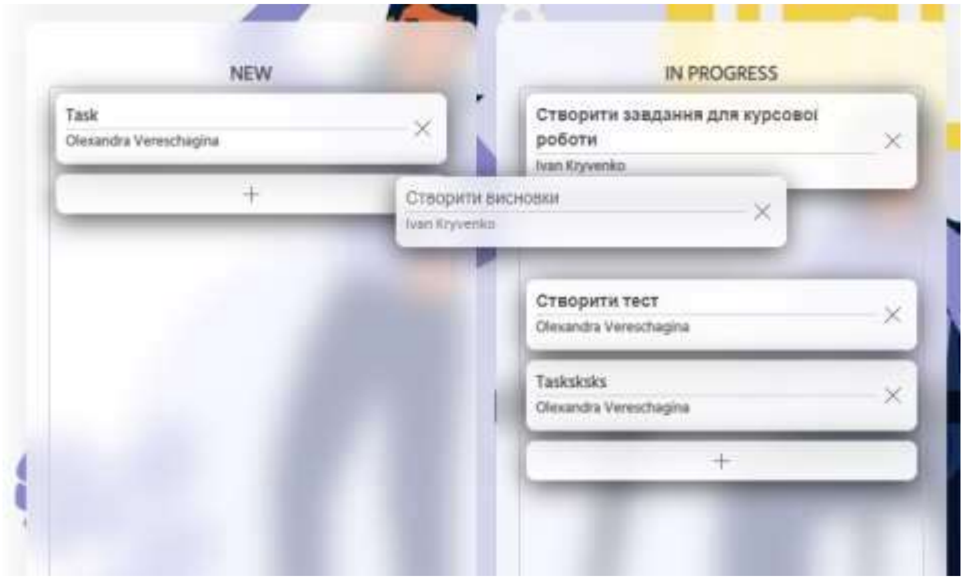


Рисунок 3.19 – Зміна статусу завдання

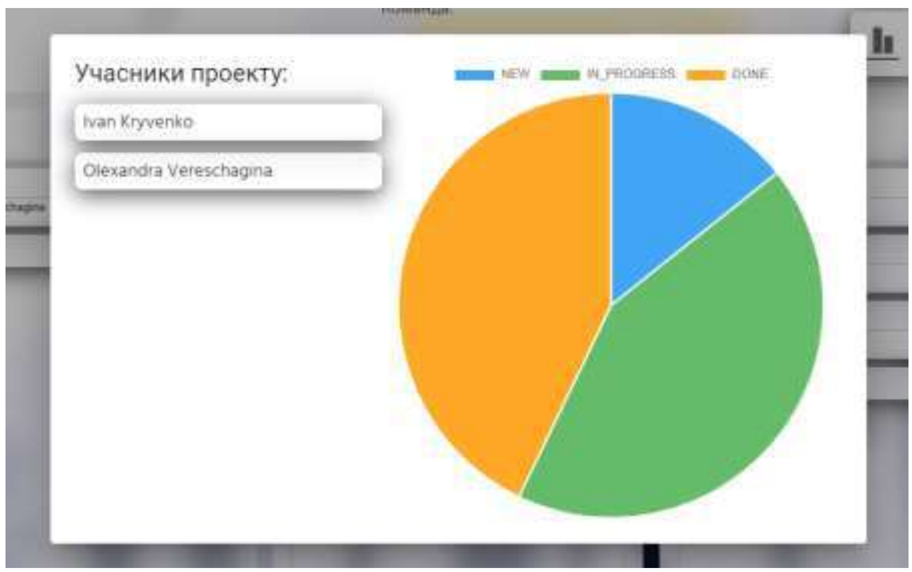


Рисунок 3.20 – Статус проекту

Розроблене програмне забезпечення володіє комплексним набором функціональних можливостей, детально описаних у табл. 3.1, які під час дослідження були детально зіставлені та порівняні з ключовими характеристиками відомих світових аналогів і сучасних систем управління проектами, таких як Jira, Trello та Microsoft Project. З метою підтвердження високої стабільності, кросплатформенності та універсальності створеного продукту, його повноцінне тестування було успішно проведене у різних операційних середовищах, включаючи

стаціонарні та мобільні операційні системи Windows, Linux, iOS, Android, а також у рамках найпопулярніших інтернет-браузерів сучасності, серед яких Chrome, FireFox, Opera та Microsoft Edge.

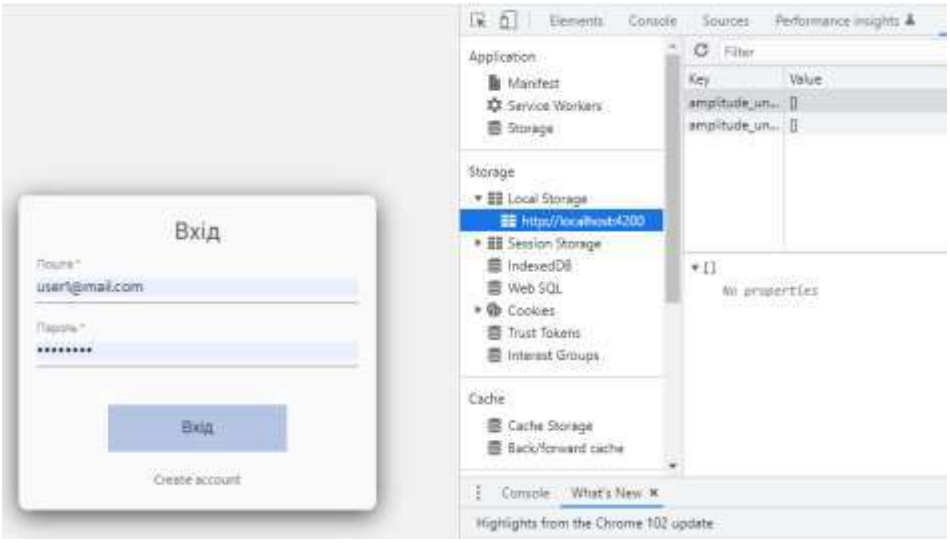


Рисунок 3.21 – Сторінка після виходу користувача

Таблиця 3.1 – Порівняльна характеристика розробленого ПЗ

Характеристика	*Jira*	*Trello*	*Microsoft*	*Easy Manage*
Можливість вибирати між різними типами	+	-	+	+
Можливість легко змінювати статус завдань проекту	+	+	+	+
Інтуїтивно зрозумілий користувацький інтерфейс	+/-	+	-	+
Можливість масштабування та розширення функціоналу	-	-	-	+
Можливість перегляду статистики по проектах	-	-	-	+

У ході фінальних випробувань було успішно проведене комплексне навантажувальне тестування системи за умови одночасної активної роботи десяти користувачів у межах п'яти різних проектів, що дозволило експериментально підтвердити повну відповідність розробленого програмного забезпечення всім початковим технічним вимогам та критеріям ефективності. При цьому зафіксований час відповіді сервера на клієнтські запити стабільно становив до 200 мс, що свідчить про високу продуктивність створеної архітектури.

Таким чином, розроблене програмне забезпечення для управління проектами пройшло всебічну перевірку на відповідність головній меті дослідження. Поточний функціонал створеної платформи є максимально розширеним, а в деяких ключових аспектах він повністю еквівалентний існуючим сучасним комерційним рішенням і цілковито задовольняє поставлене технічне завдання.

ВИСНОВКИ

У результаті написання кваліфікаційної роботи бакалавра було успішно розроблено повноцінне програмне забезпечення, яке являє собою сучасний вебдодаток для управління проектами. Усі початкові завдання, поставлені в рамках дослідження, були виконані авторами в повному обсязі завдяки послідовному аналітичному та практичному підходу. На перших етапах розробники провели глибокий аналіз наявних ринкових програмних засобів для керування процесами, що дозволило чітко обґрунтувати актуальність та необхідність створення нового продукту. У роботі детально описано ключові переваги проектування вебплатформи із залученням фреймворку Angular та нереляційної бази даних MongoDB. Орієнтуючись на сучасні стандарти дизайну UI та UX, автори спроектували оптимальну структуру та графічні компоненти системи, а також детально описали базові принципи її функціонування. Для забезпечення логічної цілісності розробники сформували загальний алгоритм роботи всієї системи та окремий алгоритм процедури створення проекту на серверній стороні.

Після детального обґрунтування вибору технологічного стеку та мов програмування було реалізовано комплексне програмне забезпечення, що включає надійні інструменти обробки інформації, маршрутизації клієнтських запитів та структурованого зберігання даних. У рамках цього процесу була створена клієнтська частина додатка з інтуїтивно зрозумілим графічним інтерфейсом користувача, яку згодом безперешкодно інтегрували з серверною стороною за допомогою спеціально спроектованого інтерфейсу API, що забезпечило повноцінне та стабільне функціонування всього вебдодатка.

Працездатність та надійність платформи були підтверджені під час навантажувального тестування, у ході якого моделювалася одночасна активна робота десяти користувачів над п'ятьма різними проектами. Результати випробувань продемонстрували високу швидкість обробки інформації із часом відповіді на запити в межах до 200 мс, що повністю відповідає початковим

технічним вимогам. Створений продукт володіє високим рівнем кросплатформенності, оскільки стабільно підтримується популярними операційними системами Windows, Linux, iOS, Android та коректно відображається у провідних інтернет-браузерах Chrome, FireFox, Opera, Microsoft Edge. Фінальні тести підтвердили, що весь розроблений функціонал працює очікувано, а поведінка системи цілковито збігається із закладеними алгоритмами. Мету було досягнуто завдяки успішному впровадженню та розширенню таких важливих функцій, як можливість гнучкого вибору між різними типами проектів, інструменти швидкої зміни поточного статусу завдань, модулі для наочного перегляду детальної статистики та загальна архітектурна здатність до легкого масштабування.

З технічної точки зору створене програмне забезпечення базується на потужному та сучасному стеку технологій. Клієнтська частина додатка, її візуальне наповнення та базова логіка на стороні користувача спроектовані за допомогою Angular. Для ефективної асинхронної обробки потоків даних та коректної взаємодії мережевих запитів із сервером була інтегрована бібліотека RxJs. Головною мовою та середовищем для написання всієї серверної логіки стали Node.js та Express.js, на базі яких розробники побудували архітектуру API та алгоритми обробки інформації. Надійне збереження даних про профілі користувачів та параметри їхніх проектів покладено на базу даних MongoDB.

Загалом створена система повністю відповідає актуальним світовим тенденціям та стандартам індустрії веб-розробки. Використані інструменти ідеально підходять для вирішення поставлених завдань, а сам додаток має чітку, прозору та послідовну архітектуру. Завдяки впровадженню компонентного підходу, де кожен елемент інтерфейсу є незалежною та інкапсульованою частиною коду, це програмне забезпечення можна безперешкодно розширювати, легко інтегруючи нові функціональні модулі та компоненти за першої потреби.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. An introduction to document-oriented Databases : website. 2022. URL: https://www.digitalocean.com/community/conceptual_articles/an-introduction-to-document-oriented-databases (дата звернення: 11.06.2026).

2. Angular – Introduction to Angular concepts : website. 2022. URL: https://angular.io/guide/architecture (дата звернення: 11.06.2026).

3. Angular – Introduction to components and templates (directives) : website. 2022. URL: https://angular.io/guide/architecture-components#directives (дата звернення: 11.06.2026).

4. Angular – Introduction to components and templates (pipes) : website. 2022. URL: https://angular.io/guide/architecture-components#pipes (дата звернення: 11.06.2026).

5. Architecture overview / Atlassian Developer : website. 2020. URL: https://developer.atlassian.com/server/jira/platform/architecture-overview/ (дата звернення: 11.06.2026).

6. Express/Node.js introduction / MDN Web Docs : website. 2022. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction (дата звернення: 11.06.2026).

7. HTTP – Overview : website. 2022. URL: https://www.tutorialspoint.com/http/http_overview.htm (дата звернення: 11.06.2026).

8. Introduction to Node.js : website. 2022. URL: https://nodejs.dev/learn (дата звернення: 11.06.2026).

9. Jira – Вікіпедія : вільна енцикл. 2022. URL: https://uk.wikipedia.org/wiki/Jira (дата звернення: 11.06.2026).

10. Manual testing for beginners : website. 2021. URL: https://www.browserstack.com/guide/manual-testing-tutorial (дата звернення: 11.06.2026).

11. Microsoft Project – Wikipedia : free encycl. 2022. URL: https://en.wikipedia.org/wiki/Microsoft_Project (дата звернення: 11.06.2026).

12. MongoDB Atlas Database : website. 2022. URL: https://www.mongodb.com/atlas/database (дата звернення: 11.06.2026).

13. UI vs. UX – What's the Difference? : website. 2018. URL: https://www.usertesting.com/blog/ui-vs-ux (дата звернення: 11.06.2026).

14. Visual Studio Code : website. 2022. URL: https://code.visualstudio.com/ (дата звернення: 11.06.2026).

15. What is an API? / AWS : website. 2022. URL: https://aws.amazon.com/what-is/api/ (дата звернення: 11.06.2026).

16. What is CRUD : website. 2022. URL: https://www.codecademy.com/article/what-is-crud (дата звернення: 11.06.2026).

17. What is frameworks? : website. 2022. URL: https://hackr.io/blog/what-is-frameworks (дата звернення: 11.06.2026).

18. What is Jira? : website. 2022. URL: https://www.productplan.com/glossary/jira/ (дата звернення: 11.06.2026).

19. What is Microsoft Project? : website. 2022. URL: https://www.projectmanager.com/blog/what-is-microsoft-project (дата звернення: 11.06.2026).

20. What is Trello? : website. 2021. URL: https://www.computerworld.com/article/3226447/what-is-trello-a-guide-to-atlassians-collaboration-and-work-management-tool.html (дата звернення: 11.06.2026).

21. What is User Interface Design? : website. 2022. URL: https://www.interaction-design.org/literature/topics/ui-design (дата звернення: 11.06.2026).

22. What is Web-Application architecture? : website. 2017. URL: https://stackify.com/web-application-architecture (дата звернення: 11.06.2026).